I. Lập và giải công thức đệ quy xác định độ phức tạp thuật toán

1)

2.4.1.a) $x_{(n)} = 2x_{(n-3)}$ for $n > 1$, $x_{(1)} = 1$

$x_{(n)} = 2x_{(n-3)}$

$= 2.2x_{(n-6)}$

$= \dots$

$= 2^{n/3}$

2.4.1.b) $x_{(n)} = x_{(n-2)} - 2$ for $n > 1$, $x_{(1)} = 0$

$x_{(n)} = x_{(n-2)} - 2$

$= (x_{(n-4)} - 2) - 2 = x_{(n-4)} - 2.2$

$= \dots$

$= x_{(1)} - (n/2).2 = -n$

2)

T(n) = $\begin{cases} 1 & khi\ n < 1 \\ T(n/2) + 1 & trong\ TH\ ngược\ lại \end{cases}$

a = 1, b = 2, d = 0

$a = b^d$ => $O(n^d \log n) = O(\log n)$

5)

a) Tổng 2^{64}-1 lần chuyển => (2^{64}-1) phút ~= $3.5.10^{13}$ năm

b) Quan sát rằng với mỗi lần di chuyển của đĩa thứ i, trước tiên, thuật toán sẽ di chuyển tháp của tất cả các đĩa nhỏ hơn nó sang một cột khác (điều này yêu cầu một lần di chuyển của đĩa thứ (i-1)) và sau đó, sau lần di chuyển thứ đĩa i, tháp nhỏ hơn này được di chuyển lên trên của nó (điều này lại yêu cầu một lần di chuyển đĩa thứ (i-1)). Như vậy, với mỗi lần di chuyển của đĩa thứ i, thuật toán di chuyển đĩa thứ (i-1) hai lần. Vì với i=1, số lần di chuyển bằng 1, chúng ta có công thức truy hồi sau cho số lần di chuyển của đĩa thứ i:

$m_{(i)} = 2m_{(i-1)}$ for $2 \leq i < n$, $m(1) = 1$

=> $m_{(i)} = 2^{i-1}$

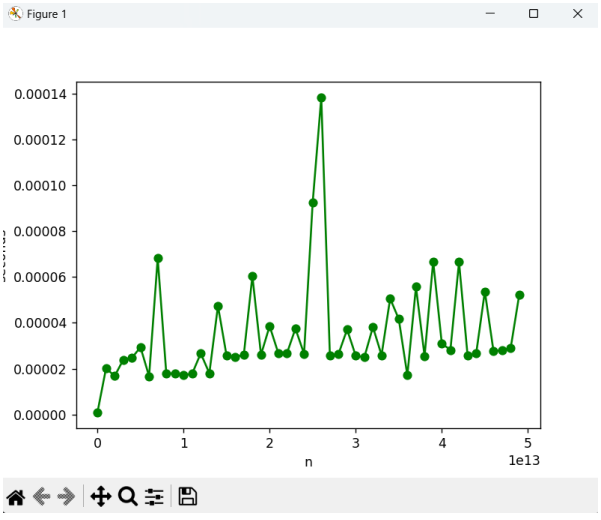
c)

```
MoveTower(n,A,B,C) :
        Start(S);
        Push((n,A,B,C,1),S);
        repeat
                while (n>1) do
                        Push((n,A,B,C,2),S);
                        n--;
                        Swap(B,C);
                endwhile;
                Move1Disk(A,B);
                Pick(S,(n,A,B,C,m))
                if (m=2):
                        Move1Disk(A,B);
                        n--;
                        Swap(A,C);
                endif
        until (m=1);
End.
```

II. Lập trình đệ quy

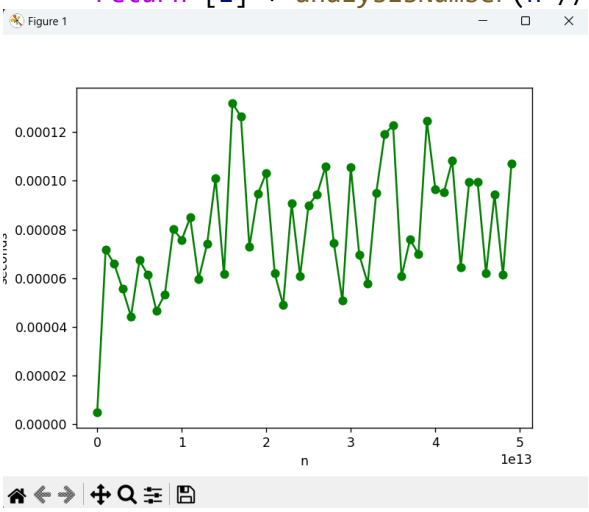
1)

```
def dec2binary(n):
    if n == 0:
        return 0
    else:
        return (n % 2 + 10 * dec2binary(int(n // 2)))
```



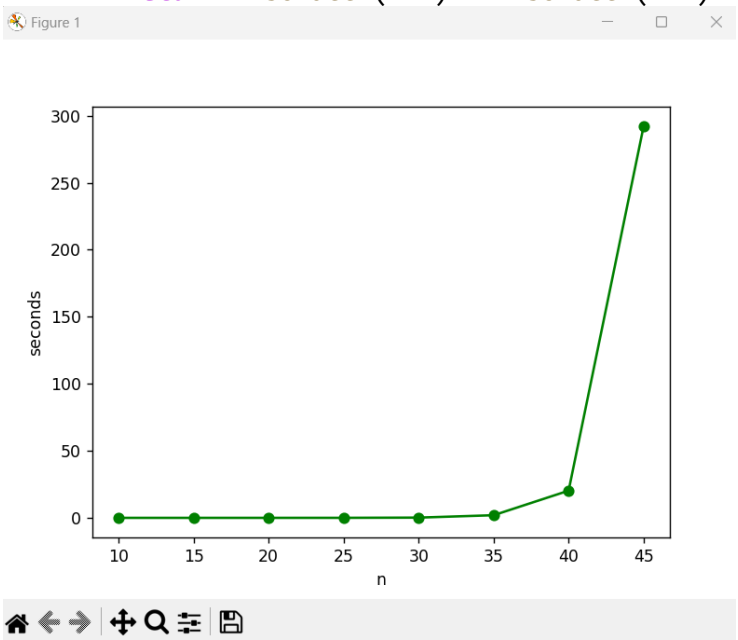
2)

```
def analysisNumber(n):
  if checkPrimeNumber(n):
    return [n]
  for i in range(2, int(math.sqrt(n))+1):
    if n % i == 0:
      return [i] + analysisNumber(n // i)
```



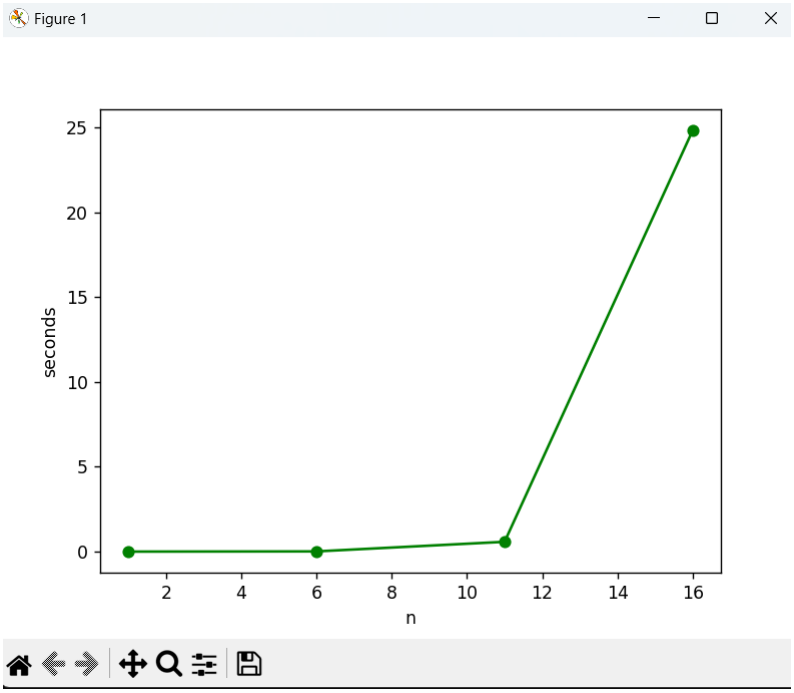
3)

```
def fibonacci(n):
    if (n == 1 or n == 2):
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```



4)

```
def MoveTower(n,A,B,C):
    if (n==1):
        Move1Dick(A,B)
    else:
        MoveTower(n-1,A,C,B)
        Move1Dick(A,B)
        MoveTower(n-1,C,B,A)
def Move1Dick(A,B):
    print(A," -> ",B)
```



III. Đặt bài toán, thiết kế, phân tích và triển khai thuật toán

Bài toán:

Bài toán chia thưởng: Có m phần thưởng được thưởng cho n học sinh giỏi có xếp hạng theo thứ tự từ 1 đến n. Hỏi có bao nhiêu cách chia các phần thưởng thoả mãn các điều kiện sau: (i) Học sinh giỏi hơn có số phần thưởng không ít hơn bạn kém hơn;

(ii) m phần thưởng phải chia hết cho các học sinh.

Lưu ý với bài này là chia mỗi học sinh ít nhất 1 phần thưởng, ai cũng xứng đáng được động viên mà ^^ Nếu không thì `for i in range(1, m + 1)` -> `for i in range(0, m + 1)`

Thuật toán:

Chia từ bạn đầu tiên (xếp hạng thấp nhất) số phần thưởng từ 1 đến m:

Trong mỗi lần chia: nếu số phần thưởng lớn hơn hoặc bằng số cuối cùng trong kết quả (số phần thưởng của bạn rank trước) thì thêm số đó vào kết quả và đệ quy với số học sinh giảm 1, số phần thưởng giảm tương ứng để tìm tiếp. Cuối cùng xóa số phần thưởng đó khỏi kết quả để tiếp tục duyệt.

Chứng minh tính đúng của bài toán:

Bước cơ sở: Ta chứng minh rằng bài toán đúng với trường hợp n = 1. Khi đó, chỉ có một học sinh giỏi nhất được nhận tất cả m phần thưởng. Số cách chia là 1.

Bước quy nạp: Ta giả sử rằng bài toán đúng với trường hợp n = k và chứng minh rằng nó cũng đúng với trường hợp n = k + 1.

+ Số cách chia m phần thưởng cho k học sinh

Khi đó, ta có thể xem xét số phần thưởng mà học sinh giỏi nhất nhận được là i (với i từ 0 đến m). Nếu i = 0, tức là học sinh giỏi nhất không nhận được phần thưởng nào, thì số cách chia cho k học sinh bằng số cách chia cho k-1 học sinh với m phần thưởng. Nếu i > 0, tức là học sinh giỏi nhất nhận được ít nhất một phần thưởng, thì số cách chia cho k học sinh bằng số cách chia cho k-1 học sinh với m - i phần thưởng và điều kiện là không ai được nhận nhiều hơn i.

+ Số cách chia m phần thưởng cho k + 1 học sinh

Nếu i = 0, tức là học sinh giỏi nhất không nhận được phần thưởng nào, thì số cách chia cho k+1 học sinh bằng số cách chia cho k học sinh với m phần thưởng. Nếu i > 0, tức là học sinh giỏi nhất nhận được ít nhất một phần thưởng, thì số cách chia cho k+1 học sinh bằng số cách chia cho k học sinh với m - i phần thưởng và điều kiện là không ai được nhận nhiều hơn i.

Suy ra đã hoàn thành việc chứng minh tính đúng của bài toán.

Độ phức tạp:

Thuật toán sử dụng 1 vòng lặp từ 1 -> m, trong mỗi vòng lặp gọi đệ quy hàm `divideGift`(m-i, n-1) để tính số cách chia cho n-1 học sinh => có độ phức tạp O(n*m)

Vậy tổng cộng độ phức tạp của thuật toán là $O(n*m^2)$

Triển khai:

```python
def divideGift(m, n, result):
    # Nếu chỉ có một học sinh thì gán tất cả các phần thưởng cho anh ta
    if n == 1:
        if m < result[-1]:
            return 0 # Trả về số cách chia là 0
        result.append(m)
        print(result)
        result.pop()
        return 1
    else:
        count = 0 # Biến lưu số cách chia
        for i in range(1, m + 1):
            # Nếu số này lớn hơn hoặc bằng số cuối cùng trong kết quả
            if not result or i >= result[-1]:
                result.append(i)
                count += divideGift(m - i, n - 1, result)
                # Xóa số này khỏi kết quả để tiếp tục duyệt
                result.pop()
        return count

# Số phần thưởng
m = 7
# Số học sinh
n = 4
# Danh sách lưu kết quả
result = []
print("Các cách chia:")
total = divideGift(m, n, result)
print("Tổng số cách:", total)
```

Các cách chia:

[1, 1, 1, 4]

[1, 1, 2, 3]

[1, 2, 2, 2]

Tổng số cách: 3